OSProj8 Designing a Virtual Memory Manager

by 潘禧辰，518021910497

Menu

Abstract

- 编写了一个Virtual Memory Manager simulator，模拟给定page number，page size，TLB entries，frame size，frame number参数下随机1000个内存访问的运行情况，统计Page-fault rate和TLB hit rate。
- 问题有两种情况，一种是frame number正好和page number相等，不需要进行页面替换；另一种是frame number比page number要少，需要进行页面替换

Environment

- Ubuntu 18.04
- Linux 5.3.0-42-generic
- VMware Workstation Rro 15.5.0 build-14665864

Quick Start

编译

Designing a Virtual Memory Manager是用户态代码，直接使用如下gcc命令进行编译

```
gcc vm.c -o vm
```

测试代码

使用以下代码测试Page-fault rate和TLB hit rate，`Physical address`和`value`的结果均存放在`output.txt`中

```
./vm addresses.txt
```

Implementation & Result

数据结构

将题目给的条件进行define

建立了以下数据结构：

- `tlb`：表示一个TLB里的block，有三个属性分别为`latest_use`用于记录上次使用时间，`page_number`记录page_number，`frame_number`记录`page_number`对应的frame_number
- `TLB`：`tlb`组成的数组，表示TLB
- `page`：表示一个页，有两个属性为`valid`即valid bit，`frame_number`记录对应的frame_number
- `page_table`：`page`组成的数组，表示页表
- `frame`：表示一个页帧，有两个属性分别为`latest_use`用于记录上次使用时间，`data`记录存储的数据
- `physical_mem`：`frame`组成的数组，表示物理内存
- `page_fault_rate`：Page-fault rate，这个变量先记录Page-fault的个数，最后除`cnt`得到结果
- `tlb_hit_rate`：TLB hit rate，这个变量先记录TLB hit的个数，最后除`cnt`得到结果
- `cnt`：用于记录当前读入了多少个地址，方便计算Page-fault rate和TLB hit rate
- `addresses`：`addresses.txt`的文件指针
- `backing_store`：`BACKING_STORE.bin`的文件指针
- `out`：`out.txt`的文件指针

```
#define PAGE_ENTRIES 256
#define PAGE_SIZE 256 // bytes
#define TLB_ENTRIES 16
#define FRAME_SIZE 256 // bytes
#define FRAME_NUM 256

typedef struct {
    int latest_use;
    int page_number;
    int frame_number;
} tlb;

typedef struct {
    bool valid;
    int frame_number;
} page;

typedef struct {
    int latest_use;
    char data[PAGE_SIZE];
} frame;

frame physical_mem[FRAME_NUM];
```

```
page page_table[PAGE_ENTRIES];
tlb TLB[TLB_ENTRIES];
float page_fault_rate = 0;
float tlb_hit_rate = 0;
int cu_time = 0;
int cnt = 0;
FILE *addresses;
FILE *backing_store;
FILE *out;
```

main

完成初始化之后定义了一些变量：

- `address`：暂时存储读入的地址
- `offset`：存储地址offset
- `page_number`：存储地址的page number
- `frame_number`：存储分配得到的frame number
- `res`：存储value
- `tlb_hit`：标记是否发生了TLB hit
- `page_fault`：标记是否发生了page fault

之后进行循环读入，每次读入一个地址，`cu_time`都要递增，方便后面进行LRU算法。读入了地址之后按照位操作提取出`offset`和`page_number`，下面依次检查几种情况：

- TLB hit：如果TLB中能够检索到对应的`page_number`那则发生了TLB hit，更新`latest_use`属性并读取`frame_number`即可
- TLB miss，page table hit：如果page table中对应`page_number`为valid，那么说明page table hit，更新`latest_use`属性，读取`frame_number`并调用`replace_tlb`替换TLB即可
- page table miss：如果继续发生page table miss，那么此时页面缺失，根据LRU选择一个合适的页帧来evict，修改其对应页表中页面的`valid`属性，调用`replace_page`完成页面替换和`latest_use`属性的更新，调用`replace_tlb`完成页面替换和`latest_use`属性的更新，读取`frame_number`

最后根据`frame_number`读取`res`即可，将该条地址的结果存入`out.txt`继续读下个地址

完成所有读取后清除文件指针，输出Page-fault rate和TLB hit rate值即可

```
int main(int argc, char *argv[]) {
    assert (argc == 2);

    for (int i = 0; i < TLB_ENTRIES; i++) {
        TLB[i].latest_use = -1;
        TLB[i].page_number = -1;
        TLB[i].frame_number = -1;
    }
    for (int i = 0; i < PAGE_ENTRIES; i++)
    {
        page_table[i].valid = false;
        page_table[i].frame_number = -1;
    }

    for (int i = 0; i < FRAME_NUM; i++)
        physical_mem[i].latest_use = -1;

    addresses = fopen(argv[1], "r");
```

```
    backing_store = fopen("BACKING_STORE.bin", "rb");
    out = fopen("output.txt","w");

    int address;
    int offset;
    int page_number;
    int frame_number;
    int res;

    bool tlb_hit;
    bool page_fault;

    fscanf(addresses, "%d", &address);
    while (!feof(addresses)) {
        cnt++;
        cu_time++;
        tlb_hit = false;
        page_fault = true;

        offset = address & 0x000000ff;
        page_number = (address >> 8) & 0x000000ff;
        //tlb hit
        for (int i = 0; i < TLB_ENTRIES; i++) {
            if (TLB[i].page_number == page_number) {
                tlb_hit = true;
                tlb_hit_rate++;
                page_fault = false;
                frame_number = TLB[i].frame_number;

                physical_mem[frame_number].latest_use = cu_time;
                TLB[i].latest_use = cu_time;
                break;
            }
        }
        //tlb miss, page table hit
        if (!tlb_hit && page_table[page_number].valid) {
            page_fault = false;
            frame_number = page_table[page_number].frame_number;
            replace_tlb(page_number, frame_number);
            physical_mem[frame_number].latest_use = cu_time;
        }
        //page table miss
        if (page_fault) {
            page_fault_rate++;
            page_table[page_number].valid = true;
            int min = INT_MAX;
            for (int i = 0; i < FRAME_NUM; i++)
                if (physical_mem[i].latest_use < min) {
                    min = physical_mem[i].latest_use;
                    frame_number = i;
                }
            for (int i = 0; i < PAGE_ENTRIES; i++)
                if (page_table[i].valid && page_table[i].frame_number ==
frame_number)
                {
                    page_table[i].valid = false;
                    break;
                }
```

```
            replace_page(page_number, frame_number);
            replace_tlb(page_number, frame_number);
            page_table[page_number].frame_number = frame_number;
        }
        res = physical_mem[frame_number].data[offset];
        fprintf(out, "Virtual address: %d Physical address: %d Value: %d\n",
address, frame_number * FRAME_SIZE + offset, res);
        fscanf(addresses, "%d", &address);
    }
    fclose(addresses);
    fclose(backing_store);
    fclose(out);
    printf("Page-fault rate: %.5f, TLB hit rate: %.5f\n", page_fault_rate / cnt,
tlb_hit_rate / cnt);
    return 0;
}
```

replace_tlb

根据`latest_use`属性选取最旧的TLB block即可，将其evicted后存入新页面，更新其属性。

```
void replace_tlb(int page_number, int frame_number) {
    int min = INT_MAX;
    int index = 0;
    for (int i = 0; i < TLB_ENTRIES; i++)
        if (TLB[i].latest_use < min) {
            min = TLB[i].latest_use;
            index = i;
        }

    TLB[index].latest_use = cu_time;
    TLB[index].page_number = page_number;
    TLB[index].frame_number = frame_number;
}
```

replace_page

使用文档中提示的`fseek`和`fread`函数完成从`BACKING_STORE.bin`中读取需要的`data`，同时更新页帧中`latest_use`属性

```
void replace_page(int page_number, int frame_number) {
    physical_mem[frame_number].latest_use = cu_time;
    fseek(backing_store, page_number * PAGE_SIZE, SEEK_SET);
    fread(physical_mem[frame_number].data, sizeof(char), PAGE_SIZE,
backing_store);
}
```

完整代码

将上述几个部分进行组合即可

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
```

```c
#include <assert.h>
#include <limits.h>

#define PAGE_ENTRIES 256
#define PAGE_SIZE 256 // bytes
#define TLB_ENTRIES 16
#define FRAME_SIZE 256 // bytes
#define FRAME_NUM 256

typedef struct {
    int latest_use;
    int page_number;
    int frame_number;
} tlb;

typedef struct {
    bool valid;
    int frame_number;
} page;

typedef struct {
    int latest_use;
    char data[PAGE_SIZE];
} frame;

void replace_tlb(int page_number, int frame_number);

void replace_page(int page_number, int frame_number);

frame physical_mem[FRAME_NUM];
page page_table[PAGE_ENTRIES];
tlb TLB[TLB_ENTRIES];
float page_fault_rate = 0;
float tlb_hit_rate = 0;
int cu_time = 0;
int cnt = 0;
FILE *addresses;
FILE *backing_store;
FILE *out;

int main(int argc, char *argv[]) {
    assert (argc == 2);

    for (int i = 0; i < TLB_ENTRIES; i++) {
        TLB[i].latest_use = -1;
        TLB[i].page_number = -1;
        TLB[i].frame_number = -1;
    }
    for (int i = 0; i < PAGE_ENTRIES; i++)
    {
        page_table[i].valid = false;
        page_table[i].frame_number = -1;
    }

    for (int i = 0; i < FRAME_NUM; i++)
        physical_mem[i].latest_use = -1;

    addresses = fopen(argv[1], "r");
```

```
    backing_store = fopen("BACKING_STORE.bin", "rb");
    out = fopen("output.txt","w");

    int address;
    int offset;
    int page_number;
    int frame_number;
    int res;

    bool tlb_hit;
    bool page_fault;

    fscanf(addresses, "%d", &address);
    while (!feof(addresses)) {
        cnt++;
        cu_time++;
        tlb_hit = false;
        page_fault = true;

        offset = address & 0x000000ff;
        page_number = (address >> 8) & 0x000000ff;
        //tlb hit
        for (int i = 0; i < TLB_ENTRIES; i++) {
            if (TLB[i].page_number == page_number) {
                tlb_hit = true;
                tlb_hit_rate++;
                page_fault = false;
                frame_number = TLB[i].frame_number;

                physical_mem[frame_number].latest_use = cu_time;
                TLB[i].latest_use = cu_time;
                break;
            }
        }
        //tlb miss, page table hit
        if (!tlb_hit && page_table[page_number].valid) {
            page_fault = false;
            frame_number = page_table[page_number].frame_number;
            replace_tlb(page_number, frame_number);
            physical_mem[frame_number].latest_use = cu_time;
        }
        //page table miss
        if (page_fault) {
            page_fault_rate++;
            page_table[page_number].valid = true;
            int min = INT_MAX;
            for (int i = 0; i < FRAME_NUM; i++)
                if (physical_mem[i].latest_use < min) {
                    min = physical_mem[i].latest_use;
                    frame_number = i;
                }
            for (int i = 0; i < PAGE_ENTRIES; i++)
                if (page_table[i].valid && page_table[i].frame_number ==
frame_number)
                {
                    page_table[i].valid = false;
                    break;
                }
```

```
            replace_page(page_number, frame_number);
            replace_tlb(page_number, frame_number);
            page_table[page_number].frame_number = frame_number;
        }
        res = physical_mem[frame_number].data[offset];
        fprintf(out, "Virtual address: %d Physical address: %d Value: %d\n",
address, frame_number * FRAME_SIZE + offset, res);
        fscanf(addresses, "%d", &address);
    }
    fclose(addresses);
    fclose(backing_store);
    fclose(out);
    printf("Page-fault rate: %.5f, TLB hit rate: %.5f\n", page_fault_rate / cnt,
tlb_hit_rate / cnt);
    return 0;
}

void replace_tlb(int page_number, int frame_number) {
    int min = INT_MAX;
    int index = 0;
    for (int i = 0; i < TLB_ENTRIES; i++)
        if (TLB[i].latest_use < min) {
            min = TLB[i].latest_use;
            index = i;
        }

    TLB[index].latest_use = cu_time;
    TLB[index].page_number = page_number;
    TLB[index].frame_number = frame_number;
}

void replace_page(int page_number, int frame_number) {
    physical_mem[frame_number].latest_use = cu_time;
    fseek(backing_store, page_number * PAGE_SIZE, SEEK_SET);
    fread(physical_mem[frame_number].data, sizeof(char), PAGE_SIZE,
backing_store);
}
```

结果

frame number == 256

Page-fault rate为0.244000，TLB hit rate为0.05500，`output.txt`与`correct.txt`中`Value`和`Physical address`均相同

```
pan@pan-virtual-machine:~/桌面/osproj/8/256$ gcc vm.c -o vm
pan@pan-virtual-machine:~/桌面/osproj/8/256$ ./vm addresses.txt
Page-fault rate: 0.24400, TLB hit rate: 0.05500
pan@pan-virtual-machine:~/桌面/osproj/8/256$ cmp output.txt correct.txt
pan@pan-virtual-machine:~/桌面/osproj/8/256$
```

frame number == 128

Page-fault rate为0.53900，TLB hit rate为0.05500，`output.txt`与`correct.txt`中`Value`相同，但是`Physical address`不同，这是因为页帧的数量不同，在frame number == 128时发生了页面替换，导致`output.txt`中使用了不同的`Physical address`

```
pan@pan-virtual-machine:~/桌面/osproj/8/128$ gcc vm.c -o vm
pan@pan-virtual-machine:~/桌面/osproj/8/128$ ./vm addresses.txt
Page-fault rate: 0.53900, TLB hit rate: 0.05500
pan@pan-virtual-machine:~/桌面/osproj/8/128$ diff output.txt correct.txt
172c172
< Virtual address: 48855 Physical address: 215 Value: -75
---
> Virtual address: 48855 Physical address: 32983 Value: -75
174c174
< Virtual address: 2035 Physical address: 755 Value: -4
---
> Virtual address: 2035 Physical address: 33267 Value: -4
176c176
< Virtual address: 14595 Physical address: 771 Value: 64
---
> Virtual address: 14595 Physical address: 33283 Value: 64
178c178
< Virtual address: 24143 Physical address: 1615 Value: -109
---
> Virtual address: 24143 Physical address: 33615 Value: -109
180c180
< Virtual address: 8113 Physical address: 1969 Value: 0
---
> Virtual address: 8113 Physical address: 33969 Value: 0
185c185
< Virtual address: 58141 Physical address: 2333 Value: 0
---
> Virtual address: 58141 Physical address: 34077 Value: 0
187,188c187,188
< Virtual address: 53040 Physical address: 3632 Value: 0
< Virtual address: 55842 Physical address: 4386 Value: 54
---
> Virtual address: 53040 Physical address: 34352 Value: 0
> Virtual address: 55842 Physical address: 34594 Value: 54
191,193c191,193
< Virtual address: 64181 Physical address: 4789 Value: 0
< Virtual address: 54879 Physical address: 5215 Value: -105
< Virtual address: 28210 Physical address: 5426 Value: 27
---
> Virtual address: 64181 Physical address: 4533 Value: 0
> Virtual address: 54879 Physical address: 3679 Value: -105
> Virtual address: 28210 Physical address: 34866 Value: 27
197,198c197,198
< Virtual address: 2149 Physical address: 5733 Value: 0
< Virtual address: 53483 Physical address: 6123 Value: 58
---
```

Difficulties

- 在编写页面替换算法时因为一开始代码顺序错误，`page_table[page_number].frame_number = frame_number` 语句放在了根据 `frame_number` 值将对应被替换掉的页面 `valid` 属性设置为 `false` 的代码前边，导致修改的是其自身页面 `valid`，修复后正常运行
- 在计算机系统结构课程中上过Cache部分，也涉及到了虚拟内存，该课程讲解的例子memory有缓存cache，一开始概念有些混乱，实现了这种结构。发现后删除cache完成修改。

Reference

- Operating System Concept 10^{th} edition
- Source code for the 10th edition of Operating System Concepts https://github.com/greggagne/osc10e